Terraform - Beginner Friendly Documentation

# 1. Introduction to Terraform

**What is Terraform?**

Terraform is an open-source tool created by HashiCorp that lets you define and manage your infrastructure using code. This approach is called Infrastructure as Code (IaC). You write configuration files that describe what your infrastructure should look like, and Terraform will create it for you.

**Why use Terraform?**

- Automates cloud infrastructure creation

- Supports many providers (AWS, Azure, GCP, etc.)

- Keeps infrastructure consistent and repeatable

- Tracks changes over time

# 2. Installing Terraform

**Step-by-step Installation**

**Windows:**

1. Go to https://developer.hashicorp.com/terraform/downloads

2. Download the Windows zip file

3. Extract the zip file to a folder, e.g., C:\terraform

4. Add that folder to your system's PATH

5. Open Command Prompt and run: terraform -v

**macOS:**

1. Use Homebrew:

brew tap hashicorp/tap

brew install hashicorp/tap/terraform

2. Check the version: terraform -v

**Linux:**

1. Download from https://developer.hashicorp.com/terraform/downloads

2. Unzip and move the binary to /usr/local/bin

3. Run: terraform -v

# 3. Configuration and Syntax

**File Structure**

Terraform uses .tf files written in **HashiCorp Configuration Language (HCL)**. Example:

* main.tf – Main configuration
* variables.tf – Declares input variables
* outputs.tf – Declares output values

**Example Folder Structure**

my-terraform-project/

main.tf # Main config: provider and resources

variables.tf # Input variables

├── outputs.tf # Output values

├── terraform.tfvars # Actual values for variables

└── README.md # Project description (optional)

**Basic Syntax**

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

}

**Explanation:**

* provider block defines which cloud service to use
* resource block defines what to create (e.g., EC2 instance)

**Variables:**

variable "region" {

default = "us-east-1"

}

**Outputs:**

output "instance\_id" {

value = aws\_instance.example.id

}

# 4. Common Terraform Commands

|  **Command** | **Description**

terraform init -Initializes the working directory

terraform plan - Shows what Terraform will do

terraform apply - Applies the changes to reach desired state

terraform destroy - Removes the resources

# 5. A Simple EC2 Example

main.tf

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "web" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

tags = {

Name = "MyWebServer"

}

}

**Steps to Run:**

terraform init

terraform plan

terraform apply

# 6. Difference Between Boto3 and Terraform

|  |  |  |
| --- | --- | --- |
| **Feature** | **Boto3** | **Terraform** |
| Type | Python SDK | Infrastructure as Code (IaC) tool |
| Language | Python | HCL (HashiCorp Configuration Language) |
| Use Case | Write Python scripts to manage AWS | Define infrastructure using code |
| Execution Style | Imperative (step-by-step instructions) | Declarative (describe the end state) |
| Speed & Control | More flexible, can add logic (if, loops) | Faster for larger infra, less logic control |
| Multi-Cloud Support | Only AWS | Supports AWS, Azure, GCP, and more |
| Community & Modules | Few reusable templates | Rich ecosystem of modules |

In short:

- Use Boto3 if you want full control with Python scripting.

- Use Terraform if you want to manage infrastructure using reusable, declarative templates.